1 Tests

For the UI, we wrote the following seven tests:

- `numPlayers` tests
  - `testGoodPlayerCount`, which tests that the UI accepts a legitimate number of players [one through twenty]
  - `testBadPlayerCount`, which tests that the UI rejects a bad number of players [negative one through negative twenty]
- Cube file parser tests
  - `testNoCubeFile`, which tests than an exception is thrown is a nonexistent file is passed to `GameManager.newGame`
  - `testEmptyCubeFile`, which tests for exceptions thrown for an empty cube file unless the game size is zero
  - `testBadCubeCount`, which tests for exceptions thrown for cube files with too few and too many cubes respectively [this is technically a design choice, to be fair, but it's better not to silently fail or truncate/extend the cubes file in our opinion]
  - `testBadFaceCount`, which tests for exceptions thrown for cube files in which cubes have too many or too few cases [also a design choice, but see what we said above]
  - `testGoodCubeCount`, which tests that good cube files are accepted

For the dictionary, wrote the following twenty-two tests:

- `ContainsTests`, which test the `contains` method, and `IsPrefixTests`, which test the `isPrefix` method [there are versions of each test listed for both `contains` and `isPrefix`]:
  - `testShortDict`, which tests whether or not a `GameDictionary` object that has loaded our custom four-word dictionary `short.txt` contains all words/all possible prefixes
  - `testNormalDict`, which does the same for `words.txt`
  - `testShortDictFail`, which tests that `contains`/`isPrefix` returns `false` for words/prefixes that should not be in the dictionary
  - `testEdgeCases`, which tests that certain special strings are not words/prefixes in the dictionary
- `LoadDictionaryTests`, which test the `loadDictionary` method:
  - `testNoFile`, which tests that `loadDictionary` fails gracefully by throwing an `IOException` when given a nonexistent file
  - `testEmptyDictionary`, which tests that a dictionary that is given an empty input file behaves as expected

  - `testNewLineDictionary`, which tests that the `loadDictionary` does not interpret trailing newlines in the input file as words
  - `testSpaceDictionary`, which does the same for spaces
  - `testInvalidDictionary`, which tests that a input file containing various invalid words [special characters, spaces, numbers, etc] is handled as expected [none of the invalid words are loaded into the dictionary but all the valid words are]
  - `testLowercase`, which tests that when loading our valid input file `short.txt`, no exceptions are thrown and the resulting dictionary contains all the words it should and none it shouldn't [the four-word dictionary]
  - `testNormalDictionary`, which does the same for your `words.txt`
- `IteratorTests`, which test the iterator returned by the `iterator` method:
  - `testEmptyHasNext`, which tests that an empty dictionary's iterator's `hasNext` method always returns false
  - `testEmptyNext`, which tests that an empty dictionary's iterator's `next` method always throws a `NoSuchElementException`
  - `testNormalHasNext`, which tests that the `hasNext` method of an iterator for a dictionary with `words.txt` loaded into it returns `true`
  - `testNormalNext`, which does the same with the `method` by checking if `"a"` or `"A"` is returned
  - `testCount`, which checks that the number of elements retrieved by iterating through the iterator of a dictionary with `short.txt` loaded into it is correct [should be four]
  - `testShortDict`, which checks that the aforementioned dictionary contains all the words it should and none that it shouldn't
  - `testNormalDict`, which does the same as the above but with `words.txt` instead of `short.txt`

2 Team 20

Team twenty failed ten of our twenty-nine tests. The specific tests failed can be found here.